Artificial Neural Networks and Deep Architecture, DD2437

SHORT REPORT

Lab Assignment 1 : Learning and generalization in feed-forward networks —from perceptron learning to backprop

Author :

Putra, Ramadhani Pamapta

Rahgozar, Parastu

Setiawan, Stanley

1. Aim and Objectives

After the lab completion, students are expected to be able to :

- Design and apply networks in classification, function approximation and generalization tasks
- Identify key limitations of single-layer networks
- Configure and monitor the behavior of learning algorithms for single- and multi-layer perceptron networks
- Recognize risks associated with backpropagation and minimize them for robust learning of multi-layer perceptrons

2. Scope

Implementation of single-layer and multi-layer perceptrons with focus on the associated learning algorithm to study their properties. On the first part of the lab assignment, students develop the code from scratch, whereas in the second part recommended libraries such as NN toolbox in Matlab can be used to examine more advanced aspects of training multi-layer perceptrons with backpropagation.

In the first part, the Delta rule (for a single-layer perceptron) and generalized Delta rule (for two-layer perceptrons) will be the mainly-focused learning algorithm. In this lab exercise, student will use the generalized delta rule (also known as 'backprop') for *classification, data compression,* and *function approximation.*

In the second part of the lab, students will work with multi-layer perceptrons to solve the problem of chaotic time series prediction. Students will design, train, validate, and evaluate the neural network with the goal to deliver a robust solution with good generalization capabilities.

3. Tools Used

- MATLAB 2017a
    - MATLAB NN Toolbox

4. Results/Findings

4.1 Lab 1 assignment - Part I

4.1.1 Classification with a single-layer perceptron

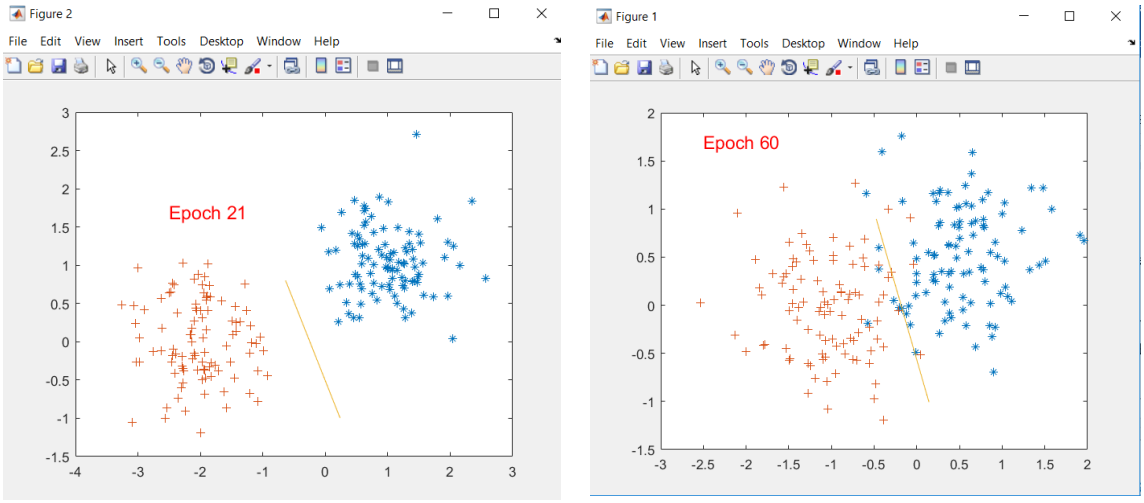


Figure 1 (left) One-Layer Neural Network with linearly separable data. (Right) One-layer neural network with non-linear separable data.

Using a single-layer perceptron, we can distinguish 2 different data distribution (or datasets) which are linearly separable. The data we use is distributed randomly as shown in Fig. 1, so our neural network can easily distinguish between those two differently-distributed data shown in picture 1(left). The number of iteration is 21 and after 21 iterations, it stopped finding the new Weight (W) value as it is already done.

However, when we tried using the same neural network with a more-closely packed separable data, the neural network seemed to have a harder time distinguishing different dataset and as a result is that it will keep on finding the new Weight(W) value until the end of iterations. In this report, we used 60 iterations as the maximum number of iteration. We tried to use more than 60 but still results similar line to distinguish the two data. Although, it can be seen that the line is located in the middle and we conclude that it separates the majority of the data despite the minor overlapping.

4.1.2 Classification and regression with a two-layer perceptron

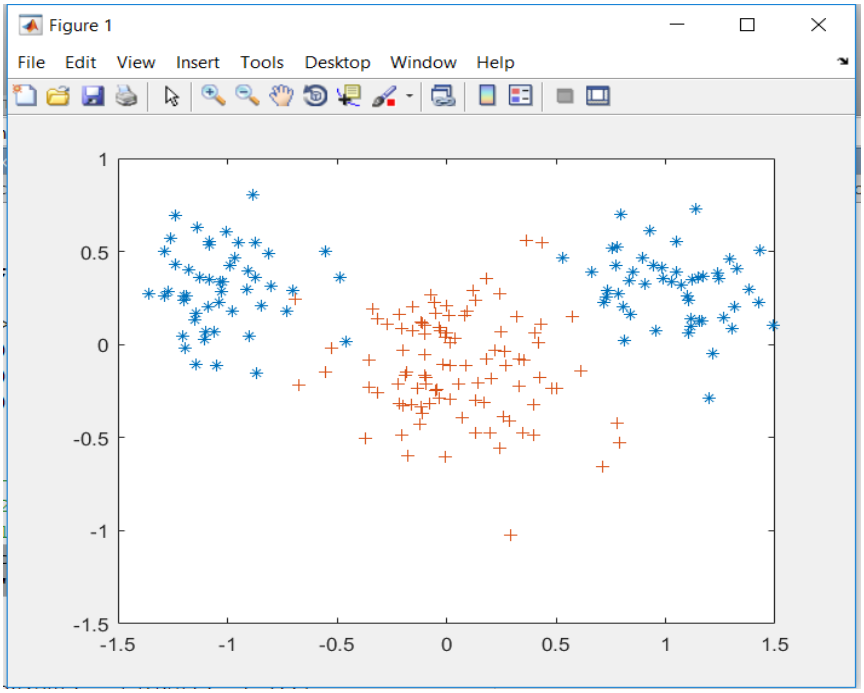


Figure 2. Non-linear separable data for two-layer neural network

When using the two-layer perceptron, we use 3 sets of data so that the neural network will show that they are able to divide the non-linear separable data as shown in figure 2. The parameters which we tried was by setting the number of hidden nodes inside the neural network.

No.	No. of hidden nodes	Iteration which error stabilized	Error Value
1.	3	197	4
2.		115	37
3.		108	3
4.	7	147	1
5.		120	7
6.		149	3
7.	15	102	5
8.		59	8
9.		97	8

Table 1. Test results from two-layered neural network

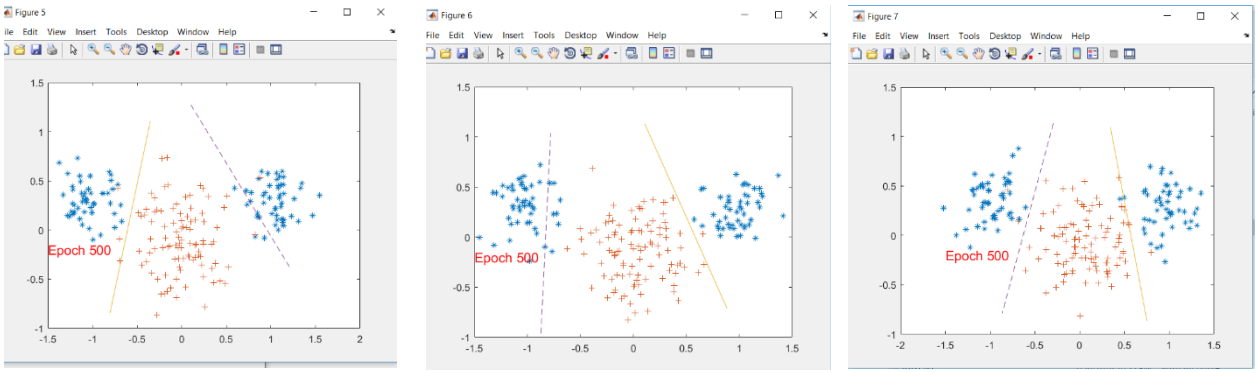


Figure 4 Data separation for 2-layer neural network. (Left) shows 3 hidden nodes, (middle) shows 7 hidden nodes, (right) shows 15 hidden nodes

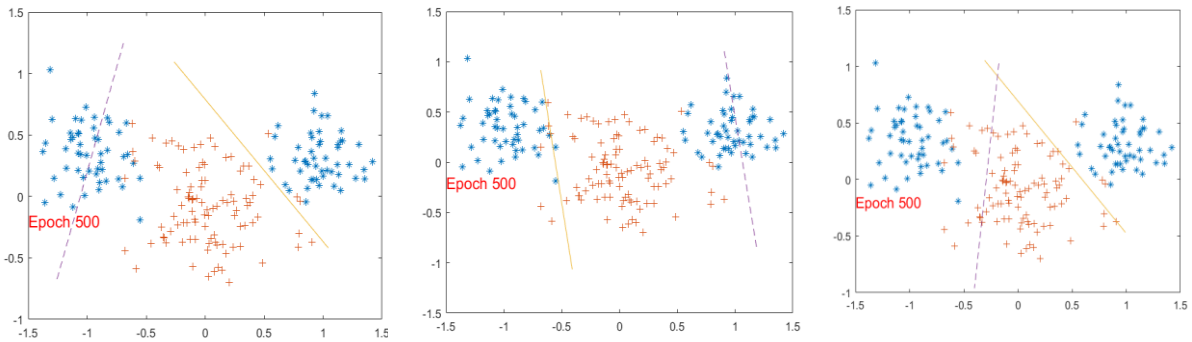


Figure 3 fixed data separation for the 2-layer neural network. (left) is 3 hidden nodes, (middle) is 7 hidden nodes and (right) is 15 hidden nodes

Table 1 shows the experiment and the data which we have tried using 3, 7 and 15 hidden nodes inside the layers. From the table, it can be seen that the mean error values for each number of nodes are the smallest for 15 number of nodes. However, error value is the least when it has 7 nodes, and the biggest when it is only 3. On the other hand, it shows a quite stable error value for 15 hidden nodes. In this report, we are going to show not all the data we have, but only some of it in Figure 3. Although the errors are quite small, we still have some trouble when trying to separate the data. We suspect it is because the number of the weight initialization as it is random, so it will be not the same in every time when we are going to train. We've managed to obtain some almost-perfect pattern separation, but this result does not always happen.

We tried a little experiment using the same datasets but with different number of hidden nodes. From the experiment, it shows that the more number of hidden nodes leads to less number of iteration to reach the stabilized error. It can be seen on Figure 4 and Table 2 in the next page.

No.	No. of hidden nodes	Iteration which error stabilized	Error Value
1.	3	337	9
2.	7	268	9
3.	15	92	9

Table 2 Test results from two-layered neural network

The Encoder Problem

```
x =

     1    -1    -1    -1    -1    -1    -1    -1
    -1     1    -1    -1    -1    -1    -1    -1
    -1    -1     1    -1    -1    -1    -1    -1
    -1    -1    -1     1    -1    -1    -1    -1
    -1    -1    -1    -1     1    -1    -1    -1
    -1    -1    -1    -1    -1     1    -1    -1
    -1    -1    -1    -1    -1    -1     1    -1
    -1    -1    -1    -1    -1    -1    -1     1
```

Figure 5. Input value for encoder problem.

In this part of the lab, we are trying to see the effect of 2-layer perceptron when forced to have a less number of nodes compared to the input. Here, we have an 8x8 input of 1 and -1 which can be seen on Figure 5. The number of layers is 2, and we set 3 hidden nodes inside the first layer. When we tried to run the program several times, it shows consistency as the number of error reaches 0 most of the time. However, there are some trials which the code

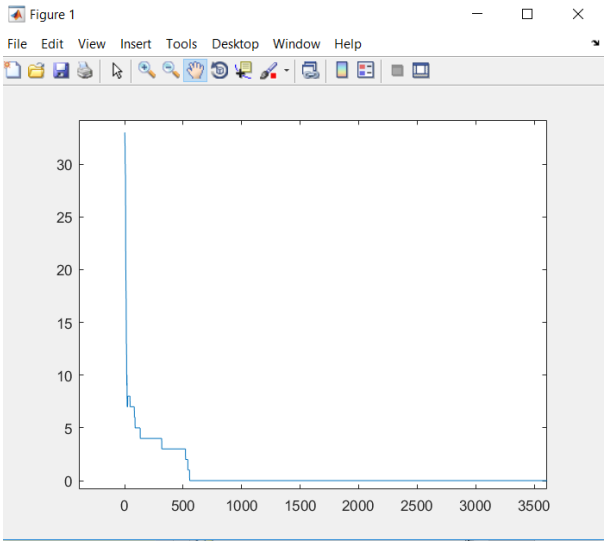


Figure 6. Error diagram for the encoder problem with bias

```
v =

   -0.7728   11.4114    5.0322   -4.8421
    4.1931    0.7644    1.0082   -3.4841
   -5.3623   -9.6007    4.6928   -5.1414
   10.3962   -5.3792    5.8360   -5.5287
    2.9557   -9.8528    5.4579   -3.9970
  -10.9608   -1.7382    4.6891   -4.9362
    7.3500    7.9667    5.5855   -4.5649
   -7.8817    6.7111    3.6253   -4.9790

w =

   -0.1752    2.8310   -0.7926    2.8598    0.2124   -3.7027    0.7559   -1.2531    1.1093
    3.8257    0.3875   -1.3523   -0.2952   -3.0394   -0.0096    1.2340    0.9727    1.2629
   -1.2898   -0.3395    3.1918    3.4774    1.7145    2.4890    1.6612    2.4623   -0.2421
```

Figure 7. The values for W and V. W is the first layer's weight and V is the second layer's weight

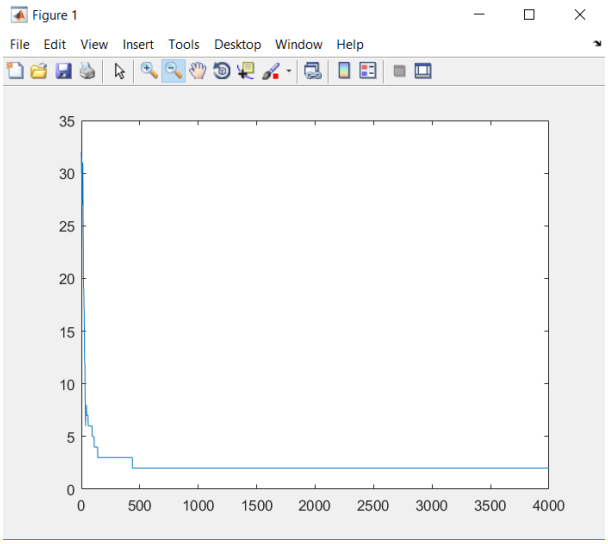


Figure 8. Error value when we do not have bias

failed to perform well, but eventually it succeeded. The error diagram is shown in figure 6. It shows that the error reaches zero after 556 iterations.

As we set the number of nodes inside the hidden layers to 3, the dimension of weight (first layer) is going to be 3x8 without the bias. Hence, when we add the bias, it will become 3x9. To compensate the first layer and to produce an 8 x 8 output, the second layer must have the size of 8x4 which will then multiplied by the output from the first layer (4x8) and thus we will have the same number of output which is 8x8. When we scrutinize inside the weights from both of the layers, we can see that the values are not the same, even though we have the same input values. Hence, we can say that by using neural network, the algorithm compensates the number nodes by adjusting the weight values, as seen on Figure 7.

The reason it can perform well is because we use **bias** inside the weights for the first and second layer. Bias inside these layers acts as a momentum for the algorithm to produce the output. When we don't have the bias, we can see that it still has some errors hence it will never produce the same value of output as the input. The error figure can be seen on Figure 8. Thus, without weights it will be impossible to make the algorithm acting as an encoder.

4.1.3 Function Approximation

Figure	Number of Steps	Number of Nodes	Number of Samples
-	500	2	1
6-a	500	8	12
-	1000	8	20
6-b	1000	20	25
-	5000	25	25
6-c	5000	25	100

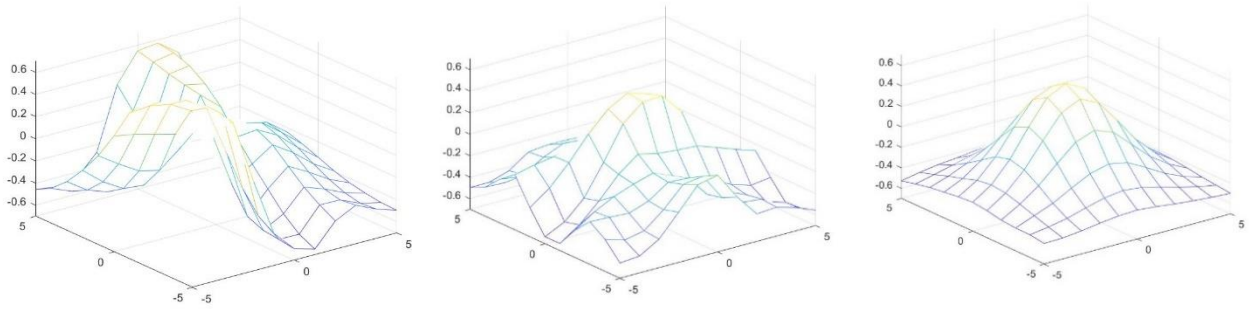
Table 3 Variated parameters in Gaussian function

In this section, unlike the previous parts, certain input and output samples will be given to the perceptron network to train based on them. To be able to observe the results visually, a 3D concept is mapped as shown in figures below.

As a known function, the "Bell shaped Gauss" function is generated inputs and outputs are randomized, some samples are extracted for training. If an animation option is used, it can be seen than after each step, the shape become more and more similar to Gaussian.

To evaluate the data generalization done by the network, we tested the network by changing 3 variables; number of Iterations (Steps), number of hidden nodes and number of samples extracted from the function. We were supposed to change the number of samples between n = 1 and n = 25. Table (3) and following figures, show the details and results.

As it is shown in the figures, the final shape is highly dependent on number of nodes and samples. Choosing less than 5 hidden nodes makes the shape lose the "Bell" figure completely,



(a) *(b)* *(c)*

Figure 9 - Shapes a,b,c with different parameters

while nodes between 5-10 affect the center of the shape, where it is dislocated from the intended center.

Moving on, after 25 nodes, it seems that a negative effect can be detected as well. So, we can conclude that when very few nodes or many nodes applied, this can ruin the final result. On the other hand, number of samples seem to have a direct relationship with the shape's orientation. As we increased the number of taken samples, the final figure resembles the bell shaped better than previous attempts.

In conclusion, we can assume that by increasing the number of epochs and hidden nodes, this may cause the "Overfitting" phenomena, and by taking more samples for training the network, results in a more precise and accurate result.

4.2 Lab 1 Assignment - Part II

Second part of this lab, is about implementing a two-layer perceptron network to a chaotic time-series prediction. In this lab assignment we use Mackey-Glass function. The desired input and output has been mentioned in the instructions. 1200 points were derived from t=301 to t=1500 to be used for training, validation and testing. 200 samples are reserved for testing.

After generating the time-series function, we need to normalize the data to fit them closer in a smaller window, and we also defined the number of epochs (1000) and a parameter to prevent overfitting.

The training was implemented with two different training regulations, Bayesian and Levenberg-Marquardt.

4.2.1 Two-Layer Perceptron

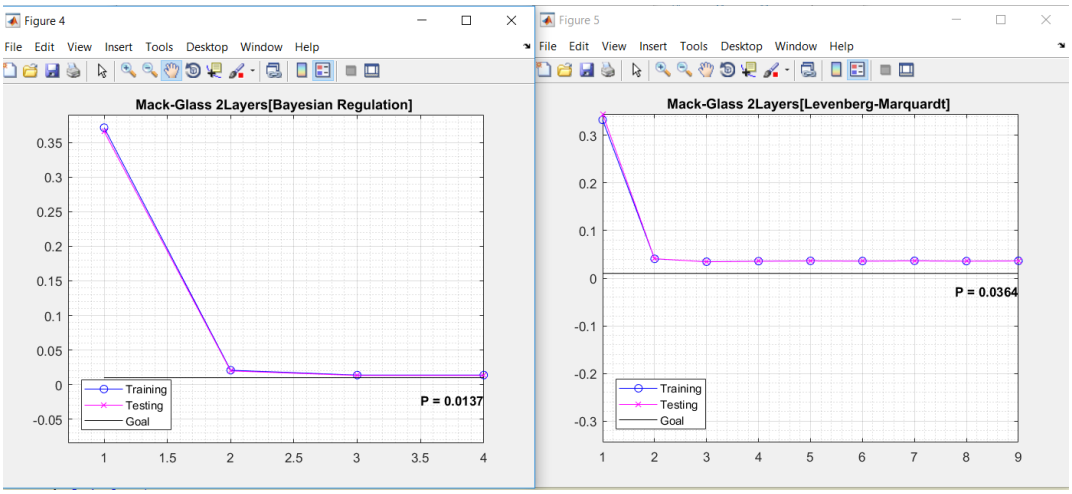


Figure 10 Comparison of using Bayesian and Levenberg-Marquardt Regulation (8 nodes)

The first task is to construct and test the 2-layer perceptron. For this part, we use Matlab's NN-toolbox. We tried using 8 hidden nodes, 0.9 regulation strength and with 2 different training regulations.

From our results in Figure 10, it can be seen that by using Bayesian regulation the training and testing are closer to the goal rather than using Levenberg-Marquardt. The output from these two different regulation methods does not differ much visually, but when we scrutinize further, there are some difference on the wave form of the output compared to the target.

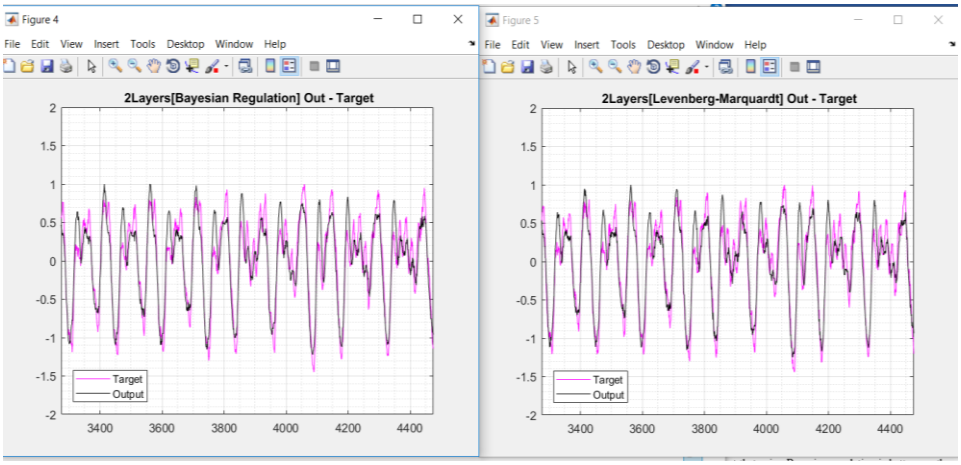


Figure 11 Comparison of Bayesian and Levenberg-Marquardt Output-Target (8 nodes)

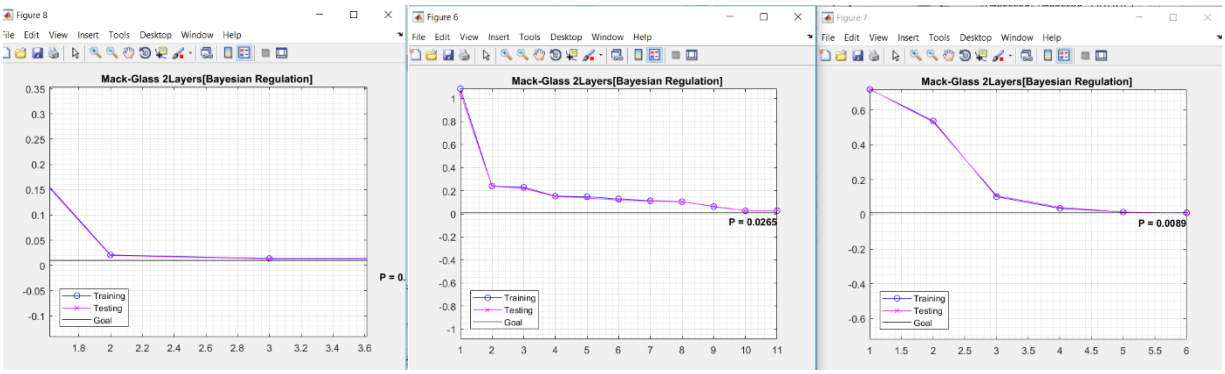


Figure 12 Comparison of the train-test and goal. Left one is the control, middle uses 0.1 regulation strength and Right uses 0.01 regulation strength

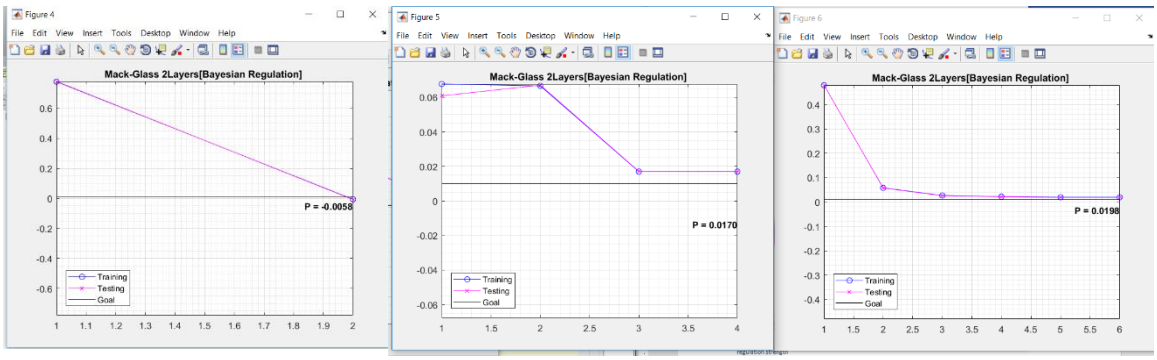


Figure 13 Comparison of the train-test and goal. Left is using 3 hidden nodes, Middle uses 1 hidden nodes and right uses 5 hidden nodes

Since we know from the experiment that using Bayesian regulation is better, we then proceed to change different nodes and regulation strength. The first parameters (8 hidden nodes and 0.9 regulation strength) are the control parameters. Next, we test using 0.1 and 0.01 regulation strength which can be seen in Figure 12. From our observation, by using 0.9 as the regulation strength, it stabilizes faster than the other two. However, using smaller values for the regulation strength makes the graph reaches goal in the later end.

Last procedure was performed by changing the number of hidden nodes. We use 8 nodes for our control (shown on Figure 9-left) and we tested using 1,3 and 5 hidden nodes in the network which can be seen on Figure 13. However, we can conclude that by using smaller number of nodes, the test and goal figure are not stable as it may change from time to time. Hence, we conclude to use larger number of hidden nodes to perform the best result.

4.2.2 Three-Layer Perceptron

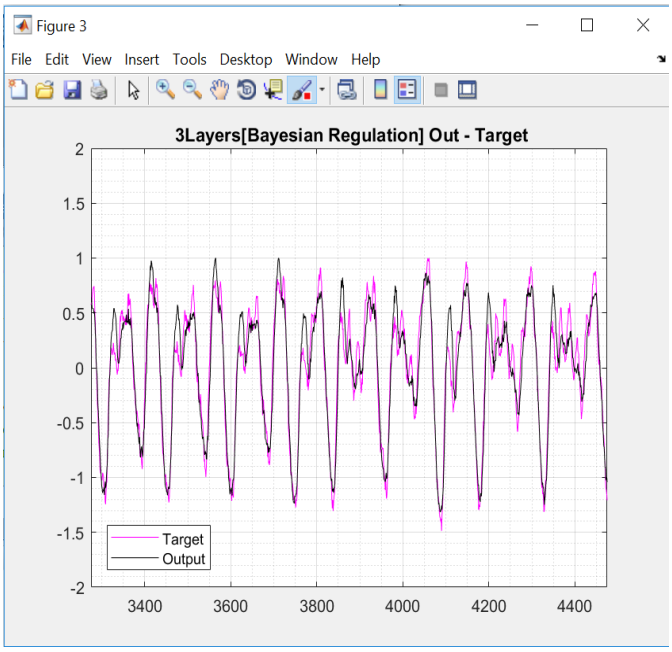


Figure 14 Bayesian regularization with 3-layer perceptron

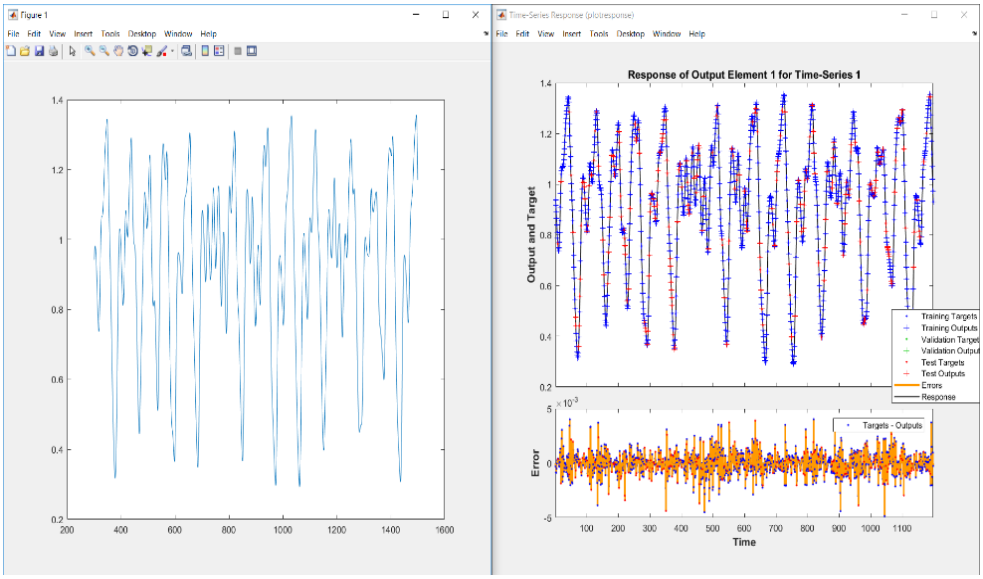


Figure 15 the figures here are similar. Left is the input reference, and right is the predicted output

In this part, we changed the features to make a three-layer perceptron with 3 nodes, which can be seen in figure X, trained with Bayesian Regularization. Moving on, we were supposed to add 'Gaussian Noise' with zero mean with different standard deviations; (0.03, 0.09, .081). To

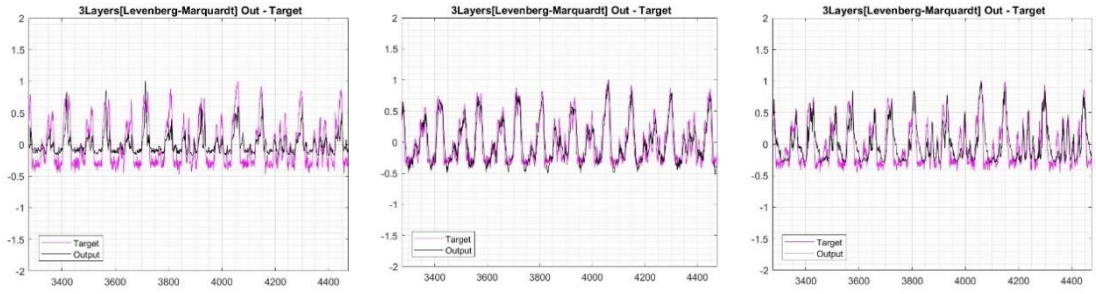


Figure 16 Results from varying the number of nodes, in presence of Gaussian noise

our conclusion, we assume that as we increase the variance, the more it worsens the training algorithm.

For more investigation, we tried to vary the number of nodes in the hidden layer in the 3-layer network. The results are shown with a constant noise variance (0.0009) and two different training rules, Bayesian and Levenberg. Number of implemented nodes are 3,8 and 12 respectively.

As it can be seen, in both figures on the left and right, the number of nodes is not sufficient for training. As a result, Output and Target signals are not fully converged on each other and will cause a considerable amount of noise in the final figure. The middle graph, has been produced using 8 nodes and gives the closest result to our satisfaction.

5. Reflection

- For the first part, it has no problem if the data is well-separated. But if it is combined with the two data, it will not separate completely (minor data overlaps).
- Using 2 layer perceptron makes it possible to distinguish the two different data. However, there are some differences when we used different number of hidden layers. But the main problem is the initialization of the weight. It will be a problem if the randomized number points out in the middle of nowhere.
- In the encoder problem case, the algorithm for 2-layer perceptron must have the capability to compensate the number of nodes for the first layer so that it will be able to determine the desired number of output, which in this case is the same as the number of input.

- In function approximation, increasing number of input samples means better accuracy and precision of the approximation. However, the number of nodes should be balanced as too few and too many nodes negatively affect the approximation.
- In the second part of the lab, we can see that by increasing the value of regulation strength, the train test seems to be stabilized faster. Increasing the number of nodes also correlates positively with the test stability.
- Adding noise in the function is assumed to cause negative impacts on the prediction accuracy.

6. Conclusion

- Increasing number of hidden layers in multilayer perceptron may result in better classification of data sets.
- Increasing nodes size and regulation strength value can help the training test to stabilize better.
- More inputs can lead to more accurate and precise prediction in function approximation